**Методические указания к лабораторной работе № 5 по курсу**

**ОСНОВЫ ПРОГРАММИРОВАНИЯ**

**ГУИМЦ**

**" Функции, макросы, рекурсия и библиотеки "**

**( 4 часа)**

Москва, МГТУ, СУЦ - 2024 год

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc525812529)

[1. 1. Цель лабораторной работы № 5 по дисциплине ОП(Основы программирования) - СУЦ 3](#_Toc525812530)

[2. 2. Порядок выполнения лабораторной работы 3](#_Toc525812531)

[3. 3. Основные понятия 3](#_Toc525812532)

[3.1. 3.1 Функции – основа процедурного программирования. 3](#_Toc525812533)

[3.2. 3.2 Понятие функции 4](#_Toc525812534)

[3.3. 3.3 Понятия, связанные с функциями в программировании 4](#_Toc525812535)

[3.4. 3.4 Описания и определения функций 5](#_Toc525812536)

[3.5. 3.5 Прототипы функций 7](#_Toc525812537)

[3.6. 3.6 Вызовы функций и возврат значений функции 7](#_Toc525812538)

[3.7. 3.7 Переменные в функциях 9](#_Toc525812539)

[3.8. 3.8 Указатели параметры и функции 9](#_Toc525812540)

[3.9. 3.9 Параметр массив в функции 9](#_Toc525812541)

[3.10. 3.10 Размещение функций в проекте 11](#_Toc525812542)

[3.11. 3.11 Рекурсивные функции 11](#_Toc525812543)

[3.12. 3.12 Макросы и переменные этапа компиляции 11](#_Toc525812544)

[3.13. 3.13 Параметры главной функции main 13](#_Toc525812545)

[3.14. 3.14 Inline функции 13](#_Toc525812546)

[3.15. 3.15 Указатели на функции 14](#_Toc525812547)

[3.16. 3.16 Библиотеки стандартных функций 14](#_Toc525812548)

[4. 4. Примеры программы с использованием функций 15](#_Toc525812549)

[4.1. 4.1 Примеры, описанные в теоретической части ЛР 15](#_Toc525812550)

[4.2. 4.2 Пузырьковая сортировка целого массива (без функции) 16](#_Toc525812551)

[4.3. 4.3 Сортировка с функцией SWAP 16](#_Toc525812552)

[5. 5. Контрольные задание ЛР №5. 17](#_Toc525812553)

[5.1. 5.1 Создать макрос 17](#_Toc525812554)

[5.2. 5.2 Функция суммы 3-х чисел 18](#_Toc525812555)

[5.3. 5.3 Функция печати массива 18](#_Toc525812556)

[5.4. 5.4 Функция Swap 18](#_Toc525812557)

[5.5. 5.5 Функция минимума или максимума по варианту 19](#_Toc525812558)

[5.6. 5.6 Функция сортировки убывание 20](#_Toc525812559)

[5.7. 5.7 Рекурсивная функция 21](#_Toc525812560)

[6. 6. Варианты заданий для студентов СУЦ. 22](#_Toc525812561)

[7. 7. Дополнительные требования для студентов СУЦ (д.т.). 22](#_Toc525812562)

[7.1. 7.1 Вложенные макросы 22](#_Toc525812563)

[7.2. 7.2 Функция экстремума с параметром типа 23](#_Toc525812564)

[7.3. 7.3 Функция сортировки, настройка типа 23](#_Toc525812565)

[7.4. 7.4 Сумма двумерного целого массива 23](#_Toc525812566)

[7.5. 7.5 Функция Swap для строк разной длины 23](#_Toc525812567)

[7.6. 7.6 Сортировка символьного массива 23](#_Toc525812568)

[7.7. 7.7 Функция с переменным числом параметров 23](#_Toc525812569)

[7.8. 7.8 Рекурсивная функция 23](#_Toc525812570)

[7.9. 7.9 Библиотеки RTL 23](#_Toc525812571)

[8. 8. Демонстрация, защита ЛР и отчет по ЛР. 23](#_Toc525812572)

[9. 9. Контрольные вопросы по ЛР. 23](#_Toc525812573)

[10. 10. Литература. 24](#_Toc525812574)

[ПРИЛОЖЕНИЯ 25](#_Toc525812575)

[First.cpp- первый модуль проекта 25](#_Toc525812576)

[Second.cpp 28](#_Toc525812577)

# 1. Цель лабораторной работы № 5 по дисциплине ОП(Основы программирования) - СУЦ

Целью данной ЛР по дисциплине ОП является получение навыков использования функций и процедур в программах. Они изучают структуру функций, способы их описания, передачи параметров и вызова функций. Так как функция является одним из главных элементов процедурного программирования, студенты учатся выделять функции в программе и проектировать более сложные программы на основе функциональной декомпозиции. Они проверяют работу отлаженных примеров и делают контрольные задания. Они выполняют отладку программы по своему варианту и получают исполнимую программу, готовую к выполнению, оформляют отчет по ЛР и защищают его.

# 2. Порядок выполнения лабораторной работы

1. Познакомиться с методическими указаниями и основными понятиями данной ЛР
2. Проработать порядок выполнения работы.
3. Создать консольные проекты для проверки примеров и выполнения задания ЛР.
4. Проверить в данном проекте примеры из методических указаний, выполнив их в отладчике в пошаговом режиме.
5. Написать программу задания ЛР по варианту, выданному преподавателем и отладить ее.
6. Продемонстрировать работу программы преподавателю в режиме отладчика по шагам и изменяемыми переменными.
7. Подготовить отчет по ЛР по представленному шаблону.
8. Защитить ЛР с предоставлением отчета и ответами на контрольные вопросы.
9. Для продвинутых студентов выполнить задания для дополнительных (необязательных) требований и также отобразить их в отчете по ЛР.

# 3. Основные понятия

В теоретической части описания лабораторной работы вводятся основные понятия и рассматриваются принципы для работы со строками на языке программирования СИ.

## 3.1 Функции – основа процедурного программирования.

Возможно, термин функция введенный в язык программирования СИ несколько собьет с толку студентов, которые усиленно изучали и изучают математику. В программировании под функцией понимается отдельно записанный фрагмент текста программы, который можно вызывать многократно, задавая разные параметры. Кроме того, для каждой функции, в зависимости от ее назначения мы можем присвоить ей имя (или название), соответствующее ее назначению. Например: **PrintArray** (печать массива) или **SortArray** (сортировка массива), что во-первых легче запоминается, а во-вторых, делает программу более наглядной и читаемой (обозримой). Последний аспект относится к понятиям абстракции действий (функций или процедур). Это позволяет, в свою очередь, абстрагироваться (отвлекаться – не учитывать) от деталей внутреннего устройства функции.

Абстракция функций или процедур является основой концепции программирования, которая называется также концепцией процедурно – ориентированного программирования (ПОП). Исторически сложилось так, что ПОП была ранее разработана и в некоторой степени повторяла работу вычислителя – компьютера, основанного на модели Машины Тьюринга. Отметим также, что концепцию ПОП иногда называют структурным подходом к программированию, что, по сути, не совсем верно. В данной лабораторной работе мы рассмотрим вопросы, связанные с: описанием и использованием функций, передачей в них параметров, вызовом функций и многие другие аспекты, непосредственно связанные с использованием функций.

В концепции процедурно ориентированного программирования главной конструкцией (модулем, строительным блоком) является процедура/функция. Разрабатываются и изучаются способы построения процедур, разделение (декомпозиция) сложной задачи на процедуры, способы связи процедур и передачи параметров между ними, совместного функционирования процедур, отладки программных систем и многое другое. Некоторые элементы концепции ПОП мы рассмотрим в этой ЛР.

## 3.2 Понятие функции

Функция – Это специально оформленный фрагмент программы, который можно вызывать по уникальному имени и настраивать на разные значения параметров.

Программа на языке программирования может быть представлена упорядоченной совокупностью последовательно расположенных операторов (**Si**):

**< S1, S2, S3, S4…, Si, Si+1, Si+2 … Sk-2 ,Sk-1, Sk>**

В предыдущих лабораторных работах мы рассмотрели вопросы циклической и разветвляющейся организации выполнения программы. Для этого используются специальные операторы цикла и ветвления. Однако их применение не дает хорошей и наглядной возможности решить проблему повторяющихся последовательностей операторов, расположенных в разных частях последовательности операторов в программе. Ниже эти повторы помечены красным:

**< S1, S2, S3, S4…, Si, Si+1, Si+2 … Sk-2 ,Sk-1, Sk>**

Эти группы операторов могут быть и большими, и, кроме того, выполняться в разных условиях с разными переменными и исходными данными. Возникает естественное желание (кстати, как в математике новое обозначение) дать им отдельное имя, однократно разместить их в отдельном месте ( **Sф** = **Si+1, Si+2** ) и обращаться к ним при необходимости. Если иметь специальный оператор вызова, с возможностью передачи параметров (**Sвф1**- вызов функции 1 - **Sф1**), то тогда нашу программу можно представить так:

**< S1, Sвф1, S4…, Si, Sвф1 … Sвф1, Sk, … > <Sф1 … >**

Размер программы при этом значительно сокращается, она становиться более наглядной, но самое главное отлаживать теперь необходимо не все группы операторов (**Si+1, Si+2**) , а только одну группу (**Sф1**), которая называется **функцией** (или процедурой). В разных языках существуют и другие названия: процедуры, подпрограммы, subroutine, методы и т.д. Названия сути не меняет.

Разбиение программ на функции при программировании дает следующие преимущества, которые, отметим, появились на стадиях эволюционного развития языков программирования:

* Функцию можно вызывать из разных мест программы, что позволяет избежать повторного программирования.
* Одну и ту же функцию можно использовать в разных программах (библиотеки).
* Использование функций повышают уровень структурированности программы и облегчают её проектирование.
* Использование функций облегчает чтение и понимание программы, ускоряет поиск и исправление ошибок.

Следствием процедурного подхода является предписание на выполнения одной главной программы – функции с названием **main**. Эта функция является главной и всегда определяет “точку входа” в программу – первый выполняемый оператор (**S1**). Функция **main** является особой, она также имеет тип, и формальные параметры, но об этом речь пойдет ниже.

## 3.3 Понятия, связанные с функциями в программировании

С функциями в языках программирования связаны следующие важные понятия:

* Определение функции или описание функции (синонимы)
* Прототип функции
* Тип возврата функции
* Формальные параметры функции
* Вызов функции
* Фактические параметры функции
* Тело функции

Эти понятия мы подробно рассмотрим ниже, а здесь начнем с простейшего работающего примера с функцией. Пусть в отдельном модуле проекта (**second.cpp**, в проекте есть еще один исходный модуль - **first.cpp**), дано описание функции суммирования двух переменных - **Summa**:

// Описание-определение функции

int Summa (int a , int b) // формальные параметры функции а и b

{

// тело функции – всего один оператор return

return (a + b); // возвращаемое значение функции типа int

};

Оператор **return** в функции задает возвращаемое функцией значение. В главной программе **main** выполнено (в исходном модуле - **first.cpp**) обращение (вызов) к этой функции непосредственно в качестве параметра функции **printf** при печати результата:

#include <stdio.h>

#include <process.h>

// Прототип функции

int Summa (int a , int b);

// Главная функция main

int main()

{

// Руссификация проекта

system ( " chcp 1251 > nul " );

//вызов функции при инициализации:

int k = Summa(1 , 1); // k=2

// Вызов функции в параметрах printf

printf ( "Сумма = %d \n" , Summa(12,13)); // фактические параметры константы

//

system(" PAUSE");

//

}

Результат работы такой программы:

**Сумма = 25**

**Для продолжения нажмите любую клавишу . . .**

## 3.4 Описания и определения функций

Термины описание функции и определение функции являются синонимами. Мы будем использовать оба этих термина. Описание функции дается однократно в каждой программе и должно быть доступно (уже известно – расположено выше по тексту) при первом вызове этой функции. При описании функции задаются:

* Название функции, уникальное имя в пределах всей программы.
* Тип возврата функции.
* Список формальных параметров функции
* Тело функции – составной оператор

Название функции это уникальное имя в программе, которое однозначно определяет действия, выполняемые данной функцией. Могут быть хорошие и плохие названия. Например, хорошие названия:

**PrintArray, FindKey, PoiskMaximum и др.**

Плохие названия функций:

**А13, Fun1, Proc2 и т.д.**

Стандартом хорошего названия являются правила так называемой “Венгерской нотации”, основной смысл которых является запись названия таким образом, что раскрывается смысл переменной или функции. Более подробно об этих правилах смотрите в литературе [10]. Формальное описание функции выглядит так:

**<Описание функции> :=[<Спецификация типа возврата функции>] <Название функции> ([<Список Формальных параметров>]) {<тело функции>};**

Спецификация типа возврата функции – это любой допустимый тип в программе: стандартный (**int, float** и т.д. ), системный (системные структуры и **typedef** переменные) и пользовательский (пользовательские структуры в СИ и классы в С++). Допускаются типы с указателями и ссылками. В отдельных случаях можно отказаться от задания типа возврата, тогда используется спецификатор – **void**. В этом случае функцию нельзя использовать в выражениях. Если спецификатор возврата отсутствует, то подразумевается возврат типа **int**.

Тело функции – это любой составной оператор, заключенный в фигурные скобки. Завершение выполнения функции производиться двумя способами:

* При достижении последней в теле функции закрывающей фигурной скобки (“}”).

При выполнении в программе специального оператора **return**.

Оператор **return** может быть задан в двух видах:

**return;** // Когда тип возврата viod

или

**return < Выражение, тип которого совпадает с типом возврата функции>;**

Список формальных параметров функции – это перечень описаний переменных со специальными именами, которые используются только в этой функции. Разделителем между описаниями является запятая. Имена формальных параметров должны быть уникальными в теле функции и хорошо подобраны по смыслу. Простейший пример описания (определения - синоним) функции:

// Описание функции суммирования

int Summa (int a , int b)

{ return (a + b);};

И вызова функции при инициализации и печати:

int k = Summa(1 , 1);

// Вызов функции как фактического параметра при печати

printf("Результат из функции = %d!\n" , Summa(5 , 5) );

Число формальных параметров не ограничивается, но не должно быть большим, для наглядности.

Пример описания функции с формальными параметрами:

int MaxMas ( int \* iMas , int Razm, int \* Max)

{

int TempMax;

TempMax = iMas[0];

for ( int i = 1; i < Razm; i++)

if ( TempMax <= iMas[i])

TempMax = iMas[i];

\*Max = TempMax;

return \*Max;

};

Параметры могут быть разных типов

// Формальные параметры разных типов

#include <stdlib.h>

…

void Func (int a, float b, int \*r, const char \* pStr)

{ \*r = (int)(a + (int)b + atoi(pStr)); }; // Тело функции

Вызов и печать:

/// Параметры

float f = 1.0f;

int Rez = 0 , a = 1;

Func (a, f, &Rez, "1");

printf("Результат из функции Func = %d!\n" , Rez );

Получим:

**Результат из функции Func = 3!**

Функция поиска максимального значения в массиве, заданном указателем (**iMas**), размерностью (**Razm**) , тип возврата **int**, возвращаемое максимальное значение указатель (**Max**).

## 3.5 Прототипы функций

Если функция должна быть вызвана в программе одного модуля (например, см. выше first.cpp), а ее описание дано а другом исходном модуле (см. **second.cpp**), необходимо задать прототип функции – краткое описание заголовка функции без ее тела. Даже при описании функции в одном исходном модуле требуется прототип, если вызов функции планируется до ее описания (оно может располагаться ниже в исходном модуле). Прототип позволяет проконтролировать правильность задания параметров при вызове функции. Прототип задается так:

**<Прототип функции> :=[<Спецификация типа возврата функции>] <Название функции> ([<Список типов параметров [с тегами]>]);**

Тело при задании прототипа функции отсутствует, вместо списка параметров задается список типов, в которых можно условно указать любые имена (они иногда называются тегами). Эти имена являются своего рода подсказками и не обязательно должны совпадать с именами формальных параметров, задаваемых в описании функции. Прототипы функции, приведенной выше, могут быть заданы так (все варианты правильные):

int MaxMas ( int \* iMas , int Razm, int \* Max); // Вариант 1

int MaxMas ( int \*, int, int \*); // Вариант 2

int MaxMas ( int \* piMas , int iRazm, int \* piMax); // Вариант 3

int MaxMas ( int \* piMas , int iRazm = 10, int \* piMax); // Вариант 4, (10) 2-й параметр по-умолчанию

Отметим на будущее, что прототип определяет так называемую сигнатуру описания функции. Если сигнатуры функций различны, например, отличается число параметров (или их типы), то функции в СИ++ могут иметь одинаковые имена. Такая технология программирования называется перегрузкой функций.

## 3.6 Вызовы функций и возврат значений функции

Вызов функции это передача управления с возвратом в тело заданной функцией и настройкой на те параметры, которые указаны при этом вызове. Эти параметры также называются фактическими. В качестве параметров , в зависимости от их типа, могут быть заданы выражения. Если в функции параметр может быть изменен, то он должен задаваться указателем. В этом случае значением выражения тоже должен быть указатель.

Формально вызов функции можно записать так:

**<Вызов функции> <Название функции> ([<Список выражений для каждого типа параметров функции>]);**

Или (чисто терминологически) можно записать так:

**<Вызов функции> <Название функции> ([<Список фактических параметров>]);**

Пример вызова функции **Summa**:

int Sum;

int a = 5 , b = 5;

Sum = Summa(2,3); // Вызов с константами

Sum = Summa(a, b); // Вызов с переменными

Sum = Summa(a, a + b); // Вызов с выражением

Sum = Summa(Summa(a,b) ,Summa(2,4)); // Вызов с вызовом функции

Пример вызова функции **MaxMas**:

// Описание массивов

int iMas[5] = {1,2,3,4,5}; // 0 - 4

int iMas1[] = {1,2,3,1,1,1,1,1}; // 0 - ?

int MaxM , c;

// Вызов Функции для массивов

c = MaxMas (iMas , sizeof(iMas)/sizeof(int) ,&MaxM);

printf ("Максимум в массиве iMas = %d \n " , c );

c = MaxMas (iMas1 , sizeof(iMas1)/sizeof(int) ,&MaxM);

printf ("Максимум в массиве = %d \n" , c );

//

Передача параметров в СИ в функцию выполняется по значению, а это означает, что в явном виде изменить параметр в функции нельзя. Например, после вызова функции (из исходного модуля проекта - **second.cpp**):

// Попытка возврата суммы через параметр sum

int Summ2 (int a , int b, int sum)

{

sum = (a + b);

return sum; // возвращаемое значение функции

};

…

В главной программе (**first.cpp**):

…

int Summ2 (int a , int b, int sum);

…

//

int SUM = 10;

Sum = Summa2(2, 3 , SUM ); // Вызов с константами

// Значение SUM = 10 , а Sum = 5 ;

Для обеспечения правильного возврата через параметр в функцию нужно передать указатель (из **second.cpp**):

int Summ3 (int a , int b, int \* psum) // Параметр указатель

{

\*psum = (a + b);

return \*psum; // возвращаемое значение функции

};

//

…

В главной программе (**first.cpp**):

…

int Summ3 (int a , int b, int \* psum);

//…

Sum= Summ3 (2 , 3, &SUM); // Вызов с указателем

// Sum= 5 и SUM = 5

В дополнение к двум рассмотренным вариантом возврата значений из функций ( через тип функции и через указатель), принципиально, можно возвратить значение и через глобальную переменную (у нас переменная **GSum**), хотя этот стиль программирования очень плохой, с позиций надежности программы. Все равно покажем пример (из **second.cpp**):

// Возврат суммы через глобальный параметр GSum

extern int GSum;

int Summ2 (int a , int b, int sum)

{

sum = (a + b);

GSum = sum; // возвращаемое значение через глобальную переменную

return sum; // возвращаемое значение функции

};

…

В главной программе (**first.cpp**):

…

int Summ2 (int a , int b, int sum);

int GSum;

…

Sum = Summa2(2, 3 , SUM ); // Вызов с константами

// Значение SUM = 10 , а Sum = 5 GSum = 5

## 3.7 Переменные в функциях

В пределах составного оператора (“тела функции”) доступны для операций, выполняемых внутри функции следующие данные:

* Формальные параметры, передаваемые при вызове функции.
* Локальные переменные, описанные в теле функции.
* Константы разного типа.
* Глобальные параметры данного исходного модуля (где описана функция) и тех, которые получены с помощью спецификатора **extern**.

Из одной функции могут быть вызваны другие функции и т.д., что позволяет спроектировать сложную иерархическую систему функций.

В функции могут быть заданы такие формальные параметры, которые не могут быть в ней изменены. Ля этого используется спецификатор **const** для формального параметра.

// Попытка изменения константного параметра “а” - const

int Summ0 (const int a , int b, int \* sum)

{

a = 5; // НА ДАННОМ ОПЕРАТОРЕ КОМПИЛЯТОР ВЫДАЕТ ОШИБКУ!!!

\*sum = (a + b);

return \*sum;

};

Формальный параметр “a” не может быть изменен в функции. Модификатор const может быть использован для характеристики всей функции, это означает, что данная функция не пожжет изменять данные объекта, но об этом вы больше узнаете в другом курсе.

## 3.8 Указатели параметры и функции

Фактические параметры передаются в функции по значению. Поэтому изменить значение переменных внутри функции невозможно!

В связи с этим, только при передаче указателя на переменную возможно их изменение. Для передачи указателя нужно:

* В описании функции для параметра задать тип указателя (\* - задан формальный параметр psum ):

int Summa (int pmas, int Razm , int **\*** **psum**){…}; // Описание функции

* И при вызове функции применить перед параметром операцию адресации/именования (& - задан формальный параметр Sum в виде указателя на переменную).

int iMas[5] = {1,2,3,4,5};

int **Sum** = 0;

int S = Summa (iMas, 5 , **&Sum**); // Вызов функции

* В самой же функции для изменения значения переменной нужно использовать операцию разименования/разадресации (\*):

**\*psum = …** // Изменение переменной по указателю в теле функции

## 3.9 Параметр массив в функции

Массив может быть передан в функцию следующими способами:

* Фиксированное число элементов в массиве (размер массива фиксирован в программе или задан глобальной переменной)
* Через указатель на массив и его размер (размер указан отдельным числовым параметром, наиболее приемлемо)
* Задание нулевого элемента в конце массива (ограниченное применение, из-за возможности нулей в массиве)
* Передача структуры параметров (указатель и массив в структуре)

В некоторых случаях размер массива – формального параметра может быть задано фиксированным (mas[5]), тогда можно воспользоваться описаниями и вызовами, представленными ниже:

// Заранее фиксировано число элементов в массиве - 5

int Summ51 (int mas[5] , int \* psum)

{

int sum = 0 ;

for (int i = 0 ; i < 5 ; i++ ) sum = sum + mas[i];

\*psum = sum ;

return \*psum; // возвращаемое значение функции

};

…

В основной программе:

int iMas[5] = {1,2,3,4,5}; // 0 - 4

printf ("Сумма в массиве iMas = %d \n" , Summ51 ( iMas, &Sum) );

..

Можно в предыдущем случае использовать для размера массива и #**define** переменные (переменные этапа компиляции).

При передаче размера массива в качестве параметров описание функции может иметь следующий вид:

// Через указатель на массив и его размер (Razm - формальный параметр)

int Summ5 (int \* mas ,int Razm , int \* psum)

{

// тело функции

int sum = 0 ;

for (int i = 0 ; i < Razm ; i++ )

sum = sum + mas[i];

\*psum = sum ;

return \*psum; // возвращаемое значение функции

};

…

В основной программе, размер массива вычисляется динамически (подчеркнуто):

int iMas[5] = {1,2,3,4,5}; // 0 - 4

printf ("Сумма в массиве iMas = %d \n" , Summ5 ( iMas, sizeof(iMas)/sizeof(int) ,&Sum) );

…

Если не предполагается в массиве хранить нулевые элементы, то в конце, массива в качестве ограничителя размера, можно поместить нуль и организовать цикл обработки до первого нуля. Роль нуля может играть и “-1”. Функция имеет вид представленный ниже, отметим, что в этом случае размер не передается.

// Нулевой элемент - конец массива

long Summ6 (int \* iMas, long \* sum)

{

int i = 0;

while (iMas[i] != 0)

{

\*sum = \*sum + iMas[i];

i++;

};

return \*sum;

}

…

В основной программе для работы алгоритма должно быть:

int iMas6[] = {1,2,3,4,5, 0}; // 0 - 4

long SumLong = 0;

printf ("Сумма в массиве iMas = %d \n" , Summ6( iMas6, &SumLong) );

Передача размерности может быть выполнена через глобальную переменную, переменную этапа компиляции. Необходимо иметь ввиду что в СИ границы индексов не контролируются. Если массив имеет несколько измерений (например, двумерный массив), то размер каждого измерения передается отдельно.

## 3.10 Размещение функций в проекте

Описание функции конкретного проекта могут быть размещены в следующих составляющих многомодульной программы:

* В этом же программном модуле в его начале (до **main**), в этом случае прототипа функции задавать не надо.
* В этом же программном модуле в его конце (после **main**), в этом случае прототип функции задавать обязательно.
* В другом исходном модуле проекта, прототип должен быть задан обязательно в либо начале главного модуля или либо в подключаемом к нему заголовочном файле.
* В подключаемом заголовочном файле, прототипа в этом случае задавать не нужно.
* В специально созданной библиотеке: \*.LIB или \*.DLL.

При невнимательном описании возможно сообщение компилятора переопределения имен (**multiply defined**). Качество проекта во многом зависит от того, как грамотно расположены функции в разных модулях и распределены для программирования по различным разработчикам, составляющим команду программистов данного проекта.

## 3.11 Рекурсивные функции

В СИ допускается использовать рекурсивные функции, которые могут вызывать сами себя. Наглядно пример рекурсивной функции можно показать при вычисление факториала натурального числа. Описание функции вычисления факториала (в математике - **n!**):

int fact( int n)

{ int rez;

if ( n == 0 )

return rez = 1;

else

return rez = n \* fact ( n - 1 ) ; };

Вызов функции вычисления факториала (**fact**) из основной программы:

//

printf ("fact 5 = %d\n" , fact(5) );

..

Полученный результат:

**fact 5 = 120**

В литературе можно познакомится и с другими вариантами рекурсивных функция. Они широко используются в алгоритмах комбинаторных вычислениях.

## 3.12 Макросы и переменные этапа компиляции

В базовом языке СИ (и, конечно, в С++) предусмотрены возможности задания макросов (макрокоманд) или переменных этапа компиляции (иногда их называют препроцессорными переменными). Для этого используется директива **#define**. Переменная этапа компиляции задается так:

**#define <имя этапа компиляции> <пробел “ ”> <выражение этапа компиляции>**

Например, для размерности массива мы можем задать переменную NMAX:

#define NMAX 10 // Описание переменной этапа компиляции

…

int iMas[NMAX]; // Использование этой переменной для задания размерности массива

…

for ( int k = 0; k < NMAX ; k++ )… // Задание числа повторений цикла

С помощью специальных директив препроцессора (#**ifndef** и #**ifdef**) можно проверить определена ли переменная этапа компиляции к данному моменту обработки текста, и вставить в программу новый фрагмент текста:

#ifndef MyLibrary

#include <my\_lib.h> // Подключение заголовочного файла

#endif

**Примечание**. Подстановка не производится в комментариях программы и текстовых константах или литералах..

При использовании макросов (макрокоманд) можно задавать параметры, на которые будет настраиваться текст макроопределения. При описании макросов задаются формальные макропараметры, которые должны быть текстовыми. Для обращения к макросам используется макровызовы, которых может быть много. Такие параметры называются фактическими макропараметрами. Определение макроса выполняется на основе следующего формального правила:

**#define <имя макроса>(<параметр>, …, <параметр>) <текст на языке, содержащий параметры>**

Имена формальных параметров должны быть уникальными в пределах описания. Между именем макроса и открывающей скобкой не должно быть пробелов. Если макрос продолжается на следующую строку текста, то используется обратная наклонная черта (“**\**”). Макровызов может быть размещен в тексте программы после определения макроса и содержит конкретные параметры. Примеры макрокоманд и макровызовов:

// Описанные макросы с параметрами

#define max(a,b) ((a>b)?a:b) // Макрос вычисления максимума их двух переменных

#define Swap(type,a,b) {type t;t=a;a=b;b=t;} // Макрос кода программы

…

// Макросы

int imax = max(3,5); // Макровызов max с константами

printf ("Максимум из двух = %d \n",imax);

// Аналогично imax = ((3>5)?3:5);

a=10 ; b = 20;

imax = max(a, b); // Макровызов max с с переменными

printf ("Максимум из двух = %d \n",imax);

// Аналогично imax = ((a>b)?a:b);

int x = 5 , y = 10 ;

printf ("До Swap x , y %d %d \n",x , y);

Swap(int,x,y);

printf ("После Swap x , y %d %d \n",x , y);

// Аналогично {int t;t=x;a=y;b=t;};

// Макрос с типом переменной

double d1 = 5.5 , d2 = 10.5 ;

printf ("До Swap d1 , d2 %f %f \n",d1 , d2);

Swap(double,d1,d2);

printf ("После Swap d1 , d2 %f %f \n",d1 , d2);

// Аналогично { double t;t=d1;a=d2;b=t;};

Результат будет таким:

**Максимум из двух = 5**

**Максимум из двух = 20**

**До Swap x , y 5 10**

**После Swap x , y 10 5**

**До Swap d1 , d2 5.500000 10.500000**

**После Swap d1 , d2 10.500000 5.500000**

**Примечание**. Использовать и разрабатывать макросы необходимо очень внимательно, так как при макроподстановке возможны различные ошибки: типов переменных, ошибки повторных описаний переменных и т.д. Такие ошибки трудно обнаружить, так как на этапе компиляции невозможно использовать отладчик. В нашем примере, если неверно указать тип переменных (вместо double задать int), выполнится неявное округление переменной и результат получиться неверным. Проверьте это на практике.

## 3.13 Параметры главной функции main

Главная функция программы может использоваться с параметрами, формат которых следующий:

void main ( int argc, [ char \* [] argv, [ char \* [] env ]]), где

**argc –** задает число параметров командной строки, если равно 1 то параметров нет.

**Argv** – массив указателей на строки представляющие параметры командной строки.

**Env** - массив указателей на строки для переменных окружения.

Небольшая программа позволяет вывести значения параметров командной строки и переменных окружения.

void main( int argc, char \* argv[] , char \* env[] )

{

…

// Число параметров командной строки

printf ("Число параметров командной строки = %d\n" , argc );

// Распечатка спика параметров

printf ("Параметры командной строки:\n" );

if ( argc >0)

{

for (int i = 0 ; i < argc ; i++)

printf ("Номер - %d Значение =%s \n" , i+1 , argv[i] );

};

// Распечатка переменных окружения (set – переменные для текущего процесса)

printf ("Переменные окружения:\n" );

int i = 0;

while ( env[i] !=NULL )

{

printf ("Номер - %d Значение =%s \n" , i+1 , env[i] );

i++;

};

Результат распечатаем не полностью, так как большой объем переменных окружения:

**Число параметров командной строки = 3**

**Параметры командной строки:**

**Номер - 1 Значение =i:\2014\_2015\kaf\оп\лр\prog\lr5\_op\debug\LR5\_OP.exe**

**Номер - 2 Значение =aaa**

**Номер - 3 Значение =ddd**

**Переменные окружения:**

**Номер - 1 Значение =ALLUSERSPROFILE=C:\Documents and Settings\All Users**

**Номер - 2 Значение =APPDATA=C:\Documents and Settings\serge\Application Data**

**… (N.B. – параметров значительно больше!)**

Первый параметр командной строки (**argv[i]**) всегда задает имя выполняемой программы, а два других мы ввели в параметры проекта: Project-> Debugging -> Comand Arguments -> aaa ddd.

## 3.14 Inline функции

В языке СИ предусмотрена возможность предписания компилятору не вызова функции (передачи управления к операторам функции), а непосредственной вставки операторов функции в текст основной программы. В ряде случаев это приводит к экономии памяти и времени выполнения программы. Такие функции называются встраиваемыми и имеют спецификатор **inline**. Такие функции обеспечивают более скоростное выполнение программы за счет оптимизации затрат на их вызов. Пример встраиваемой функции и ее использования приведен ниже:

//описание inline функция

inline int even (int x)

{

return ! (x%2); // возврат по модулю 2 четное 1 (истина) нечетное 0 (ложь)

} ;

…

Пример вызова в основной программе:

//вызов inline функции

i =10;

if (even (i)) // встраивается операция взятия по модулю с отрицание

// это эквивалентно выражению - if (!(i%2))

printf ("Число %d является четным\n", i );

else

printf ("Число %d является нечетным\n", i );

i = 5 ;

if (even (i))

printf ("Число %d является четным\n", i );

else

printf ("Число %d является нечетным\n", i );

В результате получим:

**Число 10 является четным**

**Число 5 является нечетным**

## 3.15 Указатели на функции

Опишем простые функции для демонстрации использования указателей на функции.

// Функции для указателей

//

int fun1 (int i) { return i=5;};

//

int fun2 (int i) { return i=10;};

//…

В основной программе, указатель на функцию (**pFun**) вычисляется динамически:

int i , j , k =5;

int (\* pFun) (int); // указатель на функцию с параметром int

pFun = &fun1;

i = (pFun)(k); // выражение одинаковое для вызова функции через указатель

printf ("pFun = &fun1 => %d\n" , i );

pFun = &fun2; //

j = (pFun)(k); // выражение одинаковое для вызова функции через указатель

printf ("pFun = &fun2 => %d\n" , j );

j = pFun(k); // можно и так

printf ("j = pFun(k) => %d\n" , j );

В результате получим:

**pFun = &fun1 => 5**

**pFun = &fun2 => 10**

**j = pFun(k) => 10**

## 3.16 Библиотеки стандартных функций

В системах программирования предусматривается много стандартных библиотек для функций различного назначения (например, для работы со строками, выполнения ввода и вывода, работы с массивами и т.д.). Эти библиотеки подключаются с помощью заголовочных файлов или пространств описаний (пространств имен в С++ - **namespace**). Кроме заголовочных файлов для использования библиотек подключаются специальные модули (иногда они подключаются автоматически), содержащие описания функций ( \*.lib или \*.dll). Пример подключения библиотек ввода/вывода и библиотек для работы с математическими, системными функциями и других:

#include <math.h> // Математическая библиотека функций

#include <process.h> // Системная библиотека функций

#include <string.h> // библиотека функций для работы со строками

#include <stdlib.h> // Стандартная библиотека разных функций

#include <locale.h> // библиотека локализации программ

#include <malloc.h> // библиотека динамической памяти

Стандартных библиотек и классов для описания строк очень много. Нужно хорошо знать их назначение и их состав для использования в программах. Чем лучше знания о библиотеках, тем быстрее и безошибочно можно создать сложную программу. В современных системах программирования доступны (большом количестве) библиотеки классов, которые описывают новые дополнительные типы данных. Для детального знакомства с библиотеками нужно использовать: литературу, справочники и MSDN [3] в локальном варианте (нужно инсталлировать вместе с VS) или в Интернет.

# 4. Примеры программы с использованием функций

Вторая часть задания, помимо первой связанной с изучением теоретического раздела заключается в том, чтобы испытать в проекте СИ уже отлаженные программы и фрагменты программ. Возможно, что, осваивая теоретическую часть работы, вы уже на компьютере проверили выполнение фрагментов текста и применения различных операторов ветвления (из раздела 3), тогда вам будет проще продемонстрировать их работу преподавателю. В дополнение к примерам, расположенным выше нужно испытать и изучить примеры расположенные ниже. Эти действия нужно сделать в отладчике.

Для этого нужно создать пустой проект в MS VS (Test\_LR2), как описано выше, скопировать через буфер обмена в него текст данных примеров, отладить его и выполнить.

## 4.1 Примеры, описанные в теоретической части ЛР

Нужно внимательно изучить и проверить работу всех примеров из теоретической части ЛР. Эти примеры расположены выше. Все примеры можно скопировать в свой проект. Все эти задания выполняются обязательно, они не требуют дополнительной отладки и легко (через буфер обмена -Clipboard) переносятся в программу. Все фрагменты должны демонстрироваться преподавателю. В частности, в первой части, там представлены следующие примеры:

1. Простая функция суммирования 2-х целых (**Summa**).
2. Функция максимума в целом массиве (**MaxMas**).
3. Функция с попыткой возврата значений (**Summ2**).
4. Функция с возвратом указателя (**Summ3**).
5. Функция с константным параметром (**Summ0**)
6. Функции с передачей массива в качестве параметра (**Summ51 , Summ5**)
7. Функция с массивом с нулевым элементом(**Summ6**).
8. Рекурсивная функция факториала ( **fact** ).
9. Макросы (**max** и **Swap**).
10. Пример с распечаткой параметров командной строки и окружения.
11. Пример с inline функцией.
12. Пример с вызовом функции через указатели.

Кроме этого ниже представлены примеры, которые могут быть полезными, в том числе и при выполнении контрольных заданий. Их тоже нужно изучить и проверить.

## 4.2 Пузырьковая сортировка целого массива (без функции)

Пузырьковая сортировка на основе алгоритма рассмотренного в ЛР № 3:

// Пузырьковая сортировка

int iMas[6] = {1,2,3,4,5,0}; // 0 - 5

int Razm = sizeof(iMas)/sizeof(int);

// Вывод начальный

for (int i = 0 ; i <Razm ; i++ )

printf ( "%2d \n" , iMas[ i ]);

printf ( "\n" );

int Flag = 0;

// Сортировка

for (int k= 0 ; k<Razm - 1 ;k++ )

{ Flag = 0;

for (int i = 0 ; i <Razm - k - 1 ; i++)

{ if ( iMas[ i ] > iMas[ i+1] ) // убывание

// if ( iMas[ i ] < iMas[ i+1] ) //возрастание

{ int Temp;

Temp = iMas[ i ];

iMas[ i ] = iMas[ i+1];

iMas[ i+1] = Temp;

Flag = 1;}; };

if (Flag == 0) break; };

// Вывод после сортировки

for (int i = 0 ; i <Razm ; i++ )

printf ( "%2d \n" , iMas[ i ]);

printf ( "\n" );

/////////////////////

**Результат сортировки по возрастанию:**

**1**

**2**

**3**

**4**

**5**

**0**

**0**

**1**

**2**

**3**

**4**

**5**

## 4.3 Сортировка с функцией SWAP

Выполните также использование функции **SWAP** (не путайте с макросом Swap!):

void SWAP(int \* a, int \* b)

{

int Temp;

Temp = \*a;

\*a = \*b;

\*b = Temp; };

void SWAP( int \*, int \*); // Прототип Функции

int iMas[6] = {1,2,3,4,5,0}; // 0 - 5

int Razm = sizeof(iMas)/sizeof(int);

// Вывод начальный

for (int i = 0 ; i <Razm ; i++ )

printf ( "%2d \n" , iMas[ i ]);

printf ( "\n" );

Flag = 0;

// Сортировка

for (int k= 0 ; k<Razm - 1 ;k++ )

{

Flag = 0;

for (int i = 0 ; i <Razm - k - 1 ; i++)

{

// if ( iMas[ i ] > iMas[ i+1] ) // возрастание

if ( iMas[ i ] < iMas[ i+1] ) // убывание

{ SWAP( &iMas[ i ], &iMas[ i+1] ); Flag = 1;}

};

if (Flag == 0) break;

};

// Вывод после сортировки

for (int i = 0 ; i <Razm ; i++ )

printf ( "%2d \n" , iMas[ i ]);

printf ( "\n" );

Результат сортировки по убыванию:

**0**

**1**

**2**

**3**

**4**

**5**

**5**

**4**

**3**

**2**

**1**

**0**

Нужно создать пустой проект в MS VS, как описано выше, скопировать через буфер обмена в него текст данного примера, отладить его и выполнить.

# 5. Контрольные задание ЛР №5.

## 5.1 Создать макрос

**Создать и отладить** макрос вычисления минимума или максимума их 3-х переменных. Задание уточняется вариантом. Исходные значения задаются в программе. Результат вывести на печать.

//

#define max(a,b) ((a>b)?a:b) // Макрос вычисления максимума их двух переменных

) // Макрос вычисления максимума их ТРЕХ переменных

#define max3(a,b,c) ((a>b)?((a>c)?a:((b>c)?b:c)):((b>c)?b:c))

// Вызов

// Макросы!!!!!!!!!!!!!!!!!!!

int imax = max(3,5); // Макровызов max с константами

printf ("Максимум из двух = %d \n",imax);

// Аналогично imax = ((3>5)?3:5);

a=10 ; b = 20;

imax = max(a, b); // Макровызов max с двумя переменными

printf ("Максимум из двух = %d \n",imax);

imax = max3(5, 2, 10); // Макровызов max с тремя переменными

printf ("Максимум из трех = %d \n",imax);

// Вызов

Получим Результат:

**Максимум из двух = 5**

**Максимум из двух = 20**

**Максимум из трех = 10**

## 5.2 Функция суммы 3-х чисел

**Создать и отладить функцию** сложения 3-х переменных/выражений (**Summa3**). Функцию разместить в новом модуле проекта – **second.cpp**. Результат работы функции возвращается 3-м параметром (указатель) и самой функцией. Данные заданы в программе, результат вывести на печать.

// Прототипы функции

int Summa3 (int a , int b);

int MaxMas ( int \* iMas , int Razm, int \* Max);

int Summ2 (int a , int b, int sum);

// Прототип функции

int Summa3 (int a , int b, int c, int \* psum);

// Описание

int Summa3 (int a , int b, int c, int \* psum){\*psum = (a+b+c); return \*psum;};

int Summ3 (int a , int b, int \* psum)

{

// тело функции

\*psum = (a + b);

return \*psum; // возвращаемое значение функции

};

// Вызов функции

int SUM;

int Sum= Summa3(2 , 3, 5 , &SUM); // Вызов с указателем

printf ("Сумма параметр из функции %d \n" ,SUM );

printf ("Сумма при возврате из функции %d \n" , Sum);

Получим Результат:

**Сумма параметр из функций 10**

**Сумма при возврате из функции 10**

## 5.3 Функция печати массива

**Создать функцию печати** массива (**PrintMas**). Функция описывается в другом модуле (**second.cpp**). Тип массива задан вариантом. Оформить печать красиво.

// Описание Функции Печати массива

void PrintMas ( int \* pMas, int Razm)

{

for(int i =0 ; i< Razm ; i++)

{

printf (" Элемент [%d] = %d \n" , i, pMas[i] );

};};

// Прототип функции печати массива

void PrintMas ( int \* pMas, int Razm);

// Вызов в программе

int iMas3[] = {1,2,3,4,5};

PrintMas(iMas3 , (sizeof(iMas3)/sizeof(int)));

Получим Результат:

**Элемент [0] = 1**

**Элемент [1] = 2**

**Элемент [2] = 3**

**Элемент [3] = 4**

**Элемент [4] = 5**

## 5.4 Функция Swap

**Создать и отладить функцию обмена** данными **Swap**. Функция описывается в другом модуле (**second.cpp**). Тип данных для этой функции и название зависит от варианта.

// Описание Функции обмена значениями SWAP в модуле second.cpp

void SWAP(int \* a, int \* b)

{

int Temp;

Temp = \*a;

\*a = \*b;

\*b = Temp;

};

void SWAP( int \*, int \*); // Прототип Функции

//SWAP функция ВЫЗОВ функции SWAP

int k1 = 1 , k2 = 2;

int k1 = 1 , k2 = 2;

printf ("До SWAP функции k1= %d, k2 = %d \n",k1 , k2);

SWAP(&k1 , &k2); // ВЫЗОВ функции обмена SWAP

printf ("После SWAP функции k1= %d, k2 = %d \n",k1 , k2);

Получим Результат:

**До SWAP функции k1= 1, k2 = 2**

**После SWAP функции k1= 2, k2 = 1**

## 5.5 Функция минимума или максимума по варианту

**Создать и отладить функцию поиска** в вещественном или целочисленном массиве (**float или int**) максимального или минимального значения ( **MinMas** и **MaxMas**) и запоминание его номера. Выбор алгоритма по варианту студента (см. варианты ниже). Функция описывается в другом модуле (**second.cpp**). Массив задается с помощью инициализации (10 элементов). Продемонстрировать работу алгоритма в отладчике. Массив и результат поиска распечатать в главной программе. Функцию оформить в модуле header.h, подключив его директивой **#include** в главный модуль проекта. Разработать и в отчет включить блок-схему функции.

// Максимум в массиве

int MaxMas ( int \* iMas , int Razm, int \* Max1, int \* Num)

{ int TempMax;

int NumMax=0;

TempMax = iMas[0];

for ( int i = 1; i < Razm; i++)

if ( TempMax <= iMas[i])

{ TempMax = iMas[i]; NumMax= i;}

\*Max1 = TempMax;

\*Num= NumMax;

return \*Max1; };

// Минимум в массиве

int MinMas ( int \* iMas , int Razm, int \* Min1, int \* Num)

{ int TempMin;

int NumMin=0;

TempMin = iMas[0];

for ( int i = 1; i < Razm; i++)

if ( TempMin > iMas[i])

{ TempMin = iMas[i]; NumMin= i;}

\*Min1 = TempMin;

\*Num= NumMin;

return \*Min1; };

// ПРОТОТИПЫ

int MaxMas ( int \* iMas , int Razm, int \* Max, int \*Num);

int MinMas ( int \* iMas , int Razm, int \* Max, int \*Num);

};// Вызов и печать

int Mas5[]={1,1,4,1,-2, 30,0, 7};

int NumMax = 0;

int MaxMas5 = 0;

int NumMin = 0;

int MinMas5 = 0;

MaxMas5 = 0;

MaxMas5 = MaxMas( Mas5 ,sizeof(Mas5)/sizeof(int), &MaxMas5, &NumMax);

printf ( "Max = %d Номер = %d\n" , MaxMas5, NumMax );

MaxMas5 = 0;

MinMas5 = MinMas( Mas5 ,sizeof(Mas5)/sizeof(int), & MinMas5, &NumMax);

printf ( "Min = %d Номер = %d\n" , MinMas5, NumMin );

Получим Результат:

**Max = 30 Номер = 5**

**Min = -2 Номер = 4**

## 5.6 Функция сортировки убывание

**Оформить в виде функции алгоритмы сортировки** рассмотренные выше. Варианты сортировки: пузырьковая сортировка (пуз.) и минимакс (мм.). Вид сортировки: убывание (уб.) и возрастание (воз.). Функция описывается в другом модуле (**second.cpp**). Использовать функцию **SWAP**. Тип массива определен вариантом. Для распечатки массива при проверке использовать свою функцию печати (**PrintMas**) Распечатать массив до и после сортировки. Разработать блок-схему программы и поместить в отчет.

// Функция сортировки убывание/возрастание

int SortMas(int \* iMas ,int Razm )

{ int Flag;

for (int k= 0 ; k<Razm - 1 ;k++ )

{ Flag = 0;

for (int i = 0 ; i <Razm - k - 1 ; i++)

{ // if ( iMas[ i ] > iMas[ i+1] ) // возрастание

if ( iMas[ i ] < iMas[ i+1] ) // убывание

{ SWAP( &iMas[ i ], &iMas[ i+1] ); Flag = 1;} };

if (Flag == 0) break; };

return 0; };

При вызове из главной программы:

// Прототип

int SortMas(int \* iMas ,int Razm ) ;

// Вызов сортировки

int Mas6[]={5,0, 3,10,1}; // SortMas

printf ( "ДО сортировки функция!\n" );

PrintMas (Mas6 , sizeof(Mas6)/sizeof(int));

SortMas(Mas6 , sizeof(Mas6)/sizeof(int)); // и

printf ( "После сортировки функция!\n" );

PrintMas (Mas6 , sizeof(Mas6)/sizeof(int));

Получим Результат:

**ДО сортировки функция!**

**Элемент [0] = 5**

**Элемент [1] = 0**

**Элемент [2] = 3**

**Элемент [3] = 10**

**Элемент [4] = 1**

**После сортировки функция!**

**Элемент [0] = 0**

**Элемент [1] = 3**

**Элемент [2] = 5**

**Элемент [3] = 1**

**Элемент [4] = 10**

Блок-схема функции сортировки имеет вид:

## 5.7 Рекурсивная функция

**Создать функцию вычисления** факториала целого числа с контролем на отрицательный входной параметр. В этом случае вывести значение равное нулю.

Рекурсивная функция расчета факториала число

// Рекурсивная функция факториала (n!)

int fact( int n)

{

if (n <= 0) rez = 0;

int rez;

if ( n == 0 )

return rez = 1;

else

return rez = n \* fact ( n - 1 ) ;

};

//////

При вызове из главной программы:

printf ("fact 5! = %d\n" , fact(5) );

Получим Результат:

**fact 5! = 120**

## 5.7 Функция суммирования трехмерного массива.

Создать **функцию вычисления суммы** трехмерного массива, переданного в функцию через указатель и размеры по каждому измерению. Тип массива в таблице (такой как и Swap).

## 5.7 Функция поиска максимума в диагоналях квадратной матрицы.

Создать **функцию определения максимума** в двух диагоналях квадратной матрицы, передаваемой параметром указателем и размером матрицы. Тип массива в таблице (такой как и сортировка).

# 6. Варианты заданий для студентов СУЦ.

Варианты заданий приведены ниже. Номер варианта должен соответствовать номеру студента в групповом журнале.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Создать макрос** | **Тип массива – печать и сортировка** | **Тип функции Swap** | **Поиск минимума или максимума** | **Сортировка в одномерном массиве** |
|  | Мин. Из 3-х | long | float | Максимум | Пуз. – воз. |
|  | Макс. Из 3-х | float | long | Минимум | Пуз. – уб. |
|  | Мин. Из 3-х | int | char | Максимум | Мм – воз. |
|  | Макс. Из 3-х | double | double | Минимум | Мм – уб. |
|  | Мин. Из 3-х | long | float | Максимум | Пуз. – воз. |
|  | Макс. Из 3-х | float | long | Минимум | Пуз. – уб. |
|  | Мин. Из 3-х | int | char | Максимум | Мм – воз. |
|  | Макс. Из 3-х | double | double | Минимум | Мм – уб. |
|  | Мин. Из 3-х | char | double | Максимум | Пуз. – уб. |
|  | Макс. Из 3-х | int | char | Минимум | Мм – воз. |
|  | Мин. Из 3-х | long | double | Минимум | Пуз. – уб. |

# 7. Контролируемые требования.

1. **Создание макроса 5.1**
2. **Функция суммы 3-х чисел 5.2**
3. **Функция печати массива 5.3**
4. **Функция Swap 5.4**
5. **Функция минимума или максимума 5.5**
6. **Функция сортировки** **5.6**
7. **Рекурсивная функция 5.7**
8. **Функция суммирования трехмерного массива 5.8**
9. **Функция поиска максимума в диагоналях двумерной матрицы 5.9.**

# 7. Дополнительные требования для студентов (д.т.).

Для продвинутых студентов, по желанию, можно построить программу с дополнительными требованиями. Дополнительные требования выполняются в дополнение основным требованиям ЛР. Для всех заданий нужно распечатать исходные данные и результаты, использовать второй файл и прототипы функций.

Задания дополнительных требований выполняются стедентами для тренировки и более глубокого усвоения материал. В отчет не включаются и могут быть продемонстрированы преподавателю при проверке основных требований.

## 7.1 Вложенные макросы

Создать макрос, внутри которого используются другие макросы, например вычисление максимума из двух.

## 7.2 Функция экстремума с параметром типа

Создать универсальную функцию поиска максимума и минимума и его номера. Тип (минимум или максимум) задается отдельным параметром функции.

## 7.3 Функция сортировки, настройка типа

Создать функцию сортировки массива, в которой тип сортировки задан отдельным параметром.

## 7.4 Сумма двумерного целого массива

Создать функцию вычисление суммы двумерного массива. Тип массива определяется вариантом для функции печати.

## 7.5 Функция Swap для строк разной длины

Создать функцию обмена строк разной длины. Строки заданы динамическими массивами. Нужно использовать функции библиотеки работы с динамической памятью (“кучей”).

## 7.6 Сортировка символьного массива

Создать функцию для сортировки символьного массива.

## 7.7 Функция с переменным числом параметров

Изучить технологию создания функций с переменным числом параметров. Продемонстрировать это на функции суммирования переменных.

## 7.8 Рекурсивная функция

Создать функцию вычисления факториала целого числа с контролем на отрицательный входной параметр. Контролировать также максимальное значение входного параметра и выдавать при этом сообщение. В этом случае вывести значение равное нулю.

## 7.9 Библиотеки RTL

Свести в отдельную табличку перечень библиотек СИ с пояснением основного назначения. Перечень библиотек можно получить на основе списка заголовочных файлов (\*.h и \*.hpp), размещенных в подкаталогах **INCLUDE**.

# 8. Демонстрация, защита ЛР и отчет по ЛР.

После выполнения всех необходимых шагов по ЛР, работающую программу нужно продемонстрировать преподавателю, проводящему ЛР, о чем он в журнале делает отметку. Далее студент на основе шаблона и примера оформляет отчет по ЛР. После оформления отчета, который может быть представлен преподавателю в электронном виде, выполняется защита ЛР. Студент дает ответы на вопросы по отчету и на контрольные вопросы приведенные ниже. ЛР считается полностью зачтенной, если выполнены все перечисленные требования и действия: демонстрация, отчет и защита ЛР.

# 9. Контрольные вопросы по ЛР.

1. Что такое функция в программировании?
2. Зачем нужны функции?
3. Что такое определение функции, его составляющие?
4. Что такое вызов функции, фактические параметры?
5. Что такое формальные параметры функции?
6. В чем суть концепции процедурного программирования?
7. Какой оператор используется для возврата из функций?
8. Что такое прототип функции, его составляющие и для чего он используется?
9. Как лучше задавать название функции?
10. Как можно возвратить значение данных из функций?
11. Какие переменные (данные) можно использовать в функциях?
12. Зачем используется спецификатор **extern**?
13. Что такое рекурсивные функции?
14. Что такое **inline** функции?
15. Как в функциях можно использовать глобальные и статические переменные?
16. Какие параметры можно задать для функции **main**?
17. Что такое макросы и для чего они используются?
18. Где в программе могут размещаться функции?
19. Какими способами можно передать массив в функцию?
20. Могут ли функции и макросы иметь одинаковые имена в одной программе?
21. Что такое указатели на функции и для чего они используются?
22. Какие библиотеки стандартных функций вы знаете?
23. Каково назначение оператора **return**?

# 10. Литература.

**Основная литература**

1. Список литературы, доступные книги и необходимые пособия для ЛР ОП размещены на сайте www.sergebolshakov.ruна страничке “2-й к СУЦ”. Пароль для доступа можно взять у преподавателя или старосты группы.
2. Керниган Б., Ритчи Д. К36 Язык программирования Си.\Пер. с англ., 3-е изд., испр. - СПб.: "Невский Диалект", 2001. - 352 с.: ил.
3. Касюк, С.Т. Курс программирования на языке Си: конспект лекций/С.Т. Касюк. — Челябинск: Издательский центр ЮУрГУ, 2010. — 175 с.
4. MSDN Library for Visual Studio 2005 (Vicrosoft Document Explorer – входить в состав дистрибутива VS. Нужно обязательно развернуть при установке VS VS или настроить доступ через Интернет.)
5. C.О.Бочков, Д.М.Субботин Язык программирования Си для персонального компьютера, М.: "Радио и связь", 1990.- 384 с.
6. Фридланд А.Я. Информатика и компьютерные технологии: Основные термины: Толк.слов.: Более 1000 базовых понятий и терминов. – 3-е изд., испр. и доп./ А.Я Фридланд, Л.С. Ханамирова, И.А. Фридланд – М.:ООО «Издательство Астрель»: ООО «Издательство АСТ», 2003. - 272 с.

**Дополнительная литература**

1. Общее методическое пособие по курсу для выполнения ЛР и ДЗ (см. на сайте 1-й курс [www.sergebolshakov.ru](http://www.sergebolshakov.ru)) – см. кнопку в конце каждого раздела сайта!!!
2. Другие методические материалы по дисциплине с сайта [www.sergebolshakov.ru](http://www.sergebolshakov.ru).
3. Конспекты лекций по дисциплине “Основы программирования”.
4. Подбельский В.В. Язык Си++: Учебное пособие. – М.: Финансы и статистика, 2003.
5. 5. Подбельский В.В. Стандартный Си++: Учебное пособие. – М.: Финансы и статистика, 2008.
6. Г. Шилдт “С++ Базовый курс”: Пер. с англ.- М., Издательский дом “Вильямс”, 2011 г. – 672с
7. Г. Шилдт “С++ Руководство для начинающих” : Пер. с англ. - М., Издательский дом “Вильямс”, 2005 г. – 672с
8. Г. Шилдт “Полный справочник по С++”: Пер. с англ.- М., Издательский дом “Вильямс”, 2006 г. – 800с
9. Бьерн Страуструп "Язык программирования С++"- М., Бином, 2010 г.

# ПРИЛОЖЕНИЯ

## First.cpp- первый модуль проекта

////// 3.11 Понятия ///////////// МАКРОСЫ

#define max(a,b) ((a>b)?a:b) // Макрос вычисления максимума их двух переменных

// Макрос вычисления максимума их ТРЕХ переменных

#define max3(a,b,c) ((a>b)?((a>c)?a:((b>c)?b:c)):((b>c)?b:c))

//

////// 3.3 Понятия ///////////// ФУНКЦИЯ ВОЗВРАТ

// Описание-определение простой функции

/\*

int Summa (int a , int b) // формальные параметры функции а и b

{

// тело функции – всего один оператор return

return (a + b); // возвращаемое значение функции типа int

};

\*/

/// ЗАГОЛОВОЧНЫЕ ФАЙЛЫ

#include <stdio.h>

#include <process.h>

#include <malloc.h>

//

int Summ2 (int a , int b, int sum) //

{

return (a + b);

};

int Summ3 (int a , int b, int \* psum) // Параметр указатель

{

return \*psum = (a + b); };

// Функция Печати массива

//

////// 5.7 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// РЕКУРСИВНАЯ ФУНКЦИЯ

int fact( int n)

{

int rez;

if (n <= 0) rez = 0;

if ( n == 0 )

return rez = 1;

else

return rez = n \* fact ( n - 1 ) ;

};

////// 5.3 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// ПЕЧАТЬ МАССИВА

void PrintMas ( int \* pMas, int Razm)

{

for(int i =0 ; i< Razm ; i++)

{

printf (" Элемент [%d] = %d \n" , i, pMas[i] );

};};

////// 5.5 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// МИНИМУМ/МАКСИМУМ В МАССИВЕ

// Максимум в массиве

int MaxMas ( int \* iMas , int Razm, int \* Min1, int \* Num)

{

int TempMax;

int NumMax=0;

TempMax = iMas[0];

for ( int i = 1; i < Razm; i++)

if ( TempMax > iMas[i])

{ TempMax = iMas[i]; NumMax= i;}

\*Min1 = TempMax;

\*Num= NumMax;

return \*Min1;

};

//

////// 5.5 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// МИНИМУМ/МАКСИМУМ В МАССИВЕ

// Максимум в массиве

int MinMas ( int \* iMas , int Razm, int \* Max1, int \* Num)

{

int TempMax;

int NumMax=0;

TempMax = iMas[0];

for ( int i = 1; i < Razm; i++)

if ( TempMax < iMas[i])

{ TempMax = iMas[i]; NumMax= i;}

\*Max1 = TempMax;

\*Num= NumMax;

return \*Max1;

};//

// Сортировка в массиве

////// 5.4 КОНТРОЛЬНЫЕ ЗАДАНИЯ //////

// Описание Функции обмена значениями SWAP

void SWAP(int \* a, int \* b)

{

int Temp;

Temp = \*a;

\*a = \*b;

\*b = Temp;

};

////// 5.6 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// МИНИМУМ/МАКСИМУМ В МАССИВЕ (ФУНКЦИЯ)

int SortMas(int \* iMas ,int Razm )

// Сортировка убывание

{ int Flag;

for (int k= 0 ; k<Razm - 1 ;k++ )

{

Flag = 0;

for (int i = 0 ; i <Razm - k - 1 ; i++)

{

if ( iMas[ i ] > iMas[ i+1] ) // возрастание

// if ( iMas[ i ] < iMas[ i+1] ) // убывание

{ SWAP( &iMas[ i ], &iMas[ i+1] ); Flag = 1;}

};

if (Flag == 0) break;

};

return 0;

};

//////

//////

// Прототипы функций Понятий и заданий

void SWAP( int \*, int \*); // Прототип Функции

int MaxMas ( int \* iMas , int Razm, int \* Max, int \*Num);

int MinMas ( int \* iMas , int Razm, int \* Max, int \*Num);

void PrintMas ( int \* pMas, int Razm);

///// 7.8 Понятия ///////////// ПАРАМЕТР МАССИВ

int Summ5 (int \* mas ,int Razm , int \* psum);

// ПРОТОТИПЫ ФУНКЦИЙ

int Summ2 (int a , int b, int sum);

int Summ3 (int a , int b, int \* psum); // Параметр указатель;

// ЭТА функция описана в другом файле

int Summa (int a , int b);

/////////////////////////////////

void main(void)

{

system(" chcp 1251 > nul");

//

////// 3.11 Понятия /////////////МАКРОСЫ

// Просмотр в отладчике значений !!!!!

int t1 = max(10,15);

int t3 = max3(30,10,15);

// Просмотр в отладчике значений по шагам!!!!!

////// 3.33 Понятия /////////////ФУНКЦИЯ ВОЗВРАТ ЗНАЧЕНИЯ

//ЛК5

int sum = 0;

sum= Summa (10 , 5 ); // Вызов c возвратом из функции пепосредственно

//

////// 3.33 Понятия /////////////ФУНКЦИЯ Вызов функции в качестве фактического параметра другой функции

printf ( "Сумма = %d \n" , Summa(12,13)); // фактические параметры функции для printf

//

int SUM = 0;

//

////// 3.33 Понятия /////////////ФУНКЦИЯ - ПОПЫТКА ВОЗВРАТИТЬ ЗНАЧЕНИЕ ИЗ функции в качестве параметра

//

// Просмотр в отладчике значений по шагам!!!!!

sum= Summ2 (10 , 5, SUM); // Вызов c параметром и возвратом // ОШИБКА

printf ( "Сумма возврат из функции = %d и параметром =%d \n" , sum, SUM);

////// 3.33 Понятия /////////////ФУНКЦИЯ ВОЗВРАТ С УКАЗАТЕЛЕМ

sum= Summ3 (10 , 5, &SUM);

//

///// 7.8 Понятия ///////////// ПАРАМЕТР МАССИВ в функции

int iMas[5] = {1,2,3,4,5}; // 0 - 4

PrintMas (iMas , sizeof(iMas)/sizeof(int));

printf ("Сумма в массиве iMas = %d \n" , Summ5 ( iMas, sizeof(iMas)/sizeof(int) ,&sum) );

//

////// 5.3 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////////////////////////////

////////////////////

int k1 = 1 , k2 = 2;

printf ("До SWAP функции k1 , k2 %d %d \n",k1 , k2);

SWAP(&k1 , &k2);

printf ("После SWAP функции k1 , k2 %d %d \n",k1 , k2);

//

//

int Mas5[]={1,1,4,10,-2, 30,0, 7};

////// 5.3 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// ПЕЧАТЬ МАССИВА

///////////////////////////////////////

// Печать массива

PrintMas (Mas5 , sizeof(Mas5)/sizeof(int));

// SWAP в массиве

SWAP(&Mas5[0] , &Mas5[2]);

printf ( "После Замены 0 <-> 2 !\n" );

PrintMas (Mas5 , sizeof(Mas5)/sizeof(int));

int MaxMas5 = 0;

int NumMax = 0;

////// 5.5 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// МИНИМУМ/МАКСИМУМ В МАССИВЕ

// Max и Min

MaxMas5 = MaxMas( Mas5 ,sizeof(Mas5)/sizeof(int), &MaxMas5, &NumMax);

printf ( "Max = %d Номер = %d\n" , MaxMas5, NumMax );

MaxMas5 = 0;

////// 5.5 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// МИНИМУМ/МАКСИМУМ В МАССИВЕ (ЦИКЛ)

MaxMas5 = MinMas( Mas5 ,sizeof(Mas5)/sizeof(int), &MaxMas5, &NumMax);

printf ( "Min = %d Номер = %d\n" , MaxMas5, NumMax );

// Сортировка убывание (ЦИКЛ)

int Flag; // ФЛАГ ДЛЯ ЭФФЕКТИВНОЙ СОРТИРОВКИ

int Razm = sizeof(Mas5)/sizeof(int);

// Сортировка убывание/возрвстание

////// 5.6 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// МИНИМУМ/МАКСИМУМ В МАССИВЕ (ЦИКЛ)

for (int k= 0 ; k<Razm - 1 ;k++ )

{

Flag = 0;

for (int i = 0 ; i <Razm - k - 1 ; i++)

{

// if ( iMas[ i ] > Mas5[ i+1] ) // возрастание

if ( Mas5[ i ] < Mas5[ i+1] ) // убывание

{ SWAP( &Mas5[ i ], &Mas5[ i+1] ); Flag = 1;}

};

if (Flag == 0) break;

};

printf ( "После сортировки в цикле!\n" );

// КОНЕЦ ЦИКЛА СОРТИРОВКИ

PrintMas (Mas5 , sizeof(Mas5)/sizeof(int));

//

// функция сортировки

int Mas6[]={5,0, 3,10,1}; // SortMas

printf ( "ДО сортировки функция!\n" );

////// 5.6 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// МИНИМУМ/МАКСИМУМ В МАССИВЕ (ФУНКЦИЯ)

PrintMas (Mas6 , sizeof(Mas6)/sizeof(int));

SortMas(Mas6 , sizeof(Mas6)/sizeof(int)); // и

printf ( "После сортировки функция!\n" );

PrintMas (Mas6 , sizeof(Mas6)/sizeof(int));

////// 5.7 КОНТРОЛЬНЫЕ ЗАДАНИЯ ////// РЕКУРСИВНАЯ ФУНКЦИЯ

// НУЖНО По шагам в отладчике!!

///

printf ("fact 5! = %d\n" , fact(5) );

//

system(" PAUSE");

//return 0;

//

}

## Second.cpp

////

// Описание

int Summa3 (int a , int b, int c, int \* psum){\*psum = (a+b+c); return \*psum;};

//

// Попытка изменения константного параметра “а” - const

///// 7.3 Понятия ///////////// КОНСТАНТНЫЙ ПАРАМЕТР

int Summ0 (const int a , int b, int \* sum)

{

/\*

a = 5; // НА ДАННОМ ОПЕРАТОРЕ КОМПИЛЯТОР ВЫДАЕТ ОШИБКУ!!!

\*/

\*sum = (a + b);

return \*sum;

};

///// 7.8 Понятия ///////////// ПАРАМЕТР МАССИВ

int Summ5 (int \* mas ,int Razm , int \* psum)

// Через указатель на массив и его размер (Razm - формальный параметр)

{

// тело функции

int sum = 0 ;

for (int i = 0 ; i < Razm ; i++ )

sum = sum + mas[i];

\*psum = sum ;

return \*psum; // возвращаемое значение функции

};